Day 7: No Space Left on Device

(besedilo naloge)

Podatke o direktorijih in datotekah na nekem disku dobimo v takšni obliki:

```
$ cd /
$ ls
dir a
14848514 b.txt
8504156 c.dat
dir d
$ cd a
$ ls
dir e
29116 f
2557 g
62596 h.lst
$ cd e
$ ls
584 i
$ cd ..
$ cd ..
$ cd d
$ ls
4060174 j
8033020 d.log
5626152 d.ext
7214296 k
```

V prvem delu naloge nas bo zanimala vsota velikosti vseh direktorijev, ki zasedejo manj kot 100.000 bajtov. V drugem bomo iskali direktorij, ki - skupaj s poddirektoriji -, zaseda čim manj prostora, vendar več kot toliko in toliko. Ko to vemo (ob reševanju prvega dela to sicer še ni jasno, vendar zdaj, ko to že vemo, ne zapletajmo rešitve po nepotrebnem), so pomembni le direktoriji in velikosti datotek. Zadošča, da gornji disk preberemo v

```
[14848514, 8504156, [29116, 2557, 62596, [584]], [4060174, 8033020,
5626152, 7214296]]
```

Številke so velikosti datotek; kar je v poddirektoriju, je v podseznamu.

Branje

Funkcija za branje bo takšna.

```
def read(f):
    tree = []
    for line in f:
```

```
        match line.strip().split():
            case ["$", "cd", "/"] | ["dir", _] | ["$", "ls"]:
                pass
            case ["$", "cd", ".."]:
                return tree
            case ["$", "cd", _]:
                tree.append(read(f))
            case [size, _]:
                tree.append(int(size))
    return tree
```

Razšifrirajmo. :)

Funkcija bo rekurzivna: kot argument dobi (delno prebrano) datoteko. Kot rezultat vrne seznam z vsebino trenutnega direktorija, torej seznamom z velikostmi datotek ter morebitnimi (rekurzivno pridobljenimi) seznami za poddirektorije.

V začetku funkcije pripravimo drevo.

Vsaki vrstici odluščimo končni `\n` in jo razdelimo po besedah. Nato z `match` obravnavamo različne možne vrstice.

- Vrstice `$ cd /`, `$ ls` in `dir <ime>` so nepomembne in jih preskočimo.
- Vrstica `$ cd ..` pomeni, da je trenutni direktorij prebran; vrnemo razultat.
- Ko naletimo na `$ cd <ime>`, vemo, da sledi poddirektorij. Funkcija se pokliče, da prebere njegovo vsebino, ki jo prilepi na konec drevesa.
- Zadnja možnost, je da je v vrstici dolžina datoteke in njeno (nepomembno) ime. Le-to pretvorimo v število in dodamo na konec seznama.

Stavek `match` je novost iz Pythona 3.10. V starejših različicah pač potrebujemo kup `if`-ov.

```
read(open("example.txt"))
```

```
[14848514,
 8504156,
 [29116, 2557, 62596, [584]],
 [4060174, 8033020, 5626152, 7214296]]
```

Prvi del: velikosti manjših direktorijev

Zanima nas vsota velikost vseh direktorijev, ki - skupaj s poddirektoriji - ne presežejo 100.000 bajtov. Posamični direktoriji so lahko v tej vsoti všteti tudi večkrat.

Napisati bomo morali rekurzivno funkcijo, ki ji podamo seznam, ki predstavlja direktorij, in vrača dve stvari: vsoto, ki jo zahteva naloga in skupno velikost vseh datotek v tem direktoriju. Slednjo potrebujemo zato, da bomo lahko pravilno presodili, ali je trenutni direktorij med tistimi, ki zasedejo manj kot 100.000.

Ko se enkrat odločimo, kaj bo neka rekurzivna funkcija vrnila, je ni težko napisati. :)

```
def sums(tree):
    smaller = all = 0
    for obj in tree:
        if isinstance(obj, list):
            s0, a0 = sums(obj)
            smaller += s0
            all += a0
        else:
            all += obj
    if all <= 100_000:
        smaller += all
    return smaller, all
```

`smaller` bo vsota velikost direktorijev, manjših od 100.000 (to, po čemer sprašuje naloga), `all` pa bo velikost tega direktorija in vseh pod njim.

Funkcija torej dobi seznam. Za vsak element tega seznama preveri, ali gre za seznam - torej poddirektorij. Če je tako, da pokliče samo sebe za ta poddirektorij, da dobi skupno velikost manjših direktorijev ter skupno velikost podanega direktorija. Oboje prišteje k svojima spremenljivkama. Če je element število, pa gre za datoteko, ki jo prišteje k svoji velikosti.

Ko je vsebina pregledana, preveri, ali je skupna velikost vsega, v tem direktoriju manjša od 100.000 in v tem primeru prišteje svojo velikost k skupni velikosti manjših direktorijev.

Na koncu vrne oboje - kot dogovorjeno.

```
tree = read(open("example.txt"))
smaller, all = sums(tree)
print(smaller)
```

```
95437
```

Drugi del: najmanjši dovolj velik direktorij

Kapaciteta diska je 70000000 bajtov. Potrebujemo vsaj 30000000 prostora. Pobrisali bomo en sam direktorij (s poddirektoriji), vendar čim manjšega. Katerega?

Da poenostavimo številke: ostati sme največ za 40000000 bajtov datotek, torej moramo pobrisati vsaj

```
all - 40000000
```

```
8381165
```

bajtov. Iščemo najmanjši direktorij z več kot toliko bajti. Konkretno, naloga sprašuje, koliko bajtov bomo morali pobrisati.

Napisali bomo funkcijo, ki prejme direktorij; funkcija bo seveda rekurzivna, torej bo prejela direktorij kjerkoli znotraj te hierarhije direktorijev. Drugi argument bo število bajtov, ki jih je potrebno pobrisati. Vrnila bo minimalno število bajtov, ki jih bo treba pobrisati. Če celo brisanje celotnega direktorija ne doseže meje, pa bo vrnila kar velikost tega direktorij.

```
def closest_to(tree, required):
    recommended = [closest_to(obj, required) for obj in tree if isinstance(obj, dict)]
    sufficient = [x for x in recommended if x >= required]
    if sufficient:
        return min(sufficient)
    return sum(recommended) + sum(obj for obj in tree if isinstance(obj, int))
```

Najprej gremo prek vseh poddirektorijev (`for obj in tree if isinstance(obj, dict)`) in zberemo njihove "predloge". Ti so to, kar vrača ta funkcija, torej število bajtov, ki jih bomo pobrisali v poddirektoriju (ali enem njegovih poddirektorijev); če brisanje nekega poddirektorija ne zadošča, pa je v tem seznamu velikost tega poddirektorija. To shranimo v `recommended`.

Nato naberemo vse elemente `recommended`, katerih brisanje zadošča. Če ta seznam ni prazen, vrnemo njegov najmanjši element. Sicer pa vrnemo vsoto velikosti poddirektorijev in še vseh datotek v tem direktoriju.

```
closest_to(tree, all - 40000000)
```

```
23352670
```

Kodo se da še nekoliko skrajšati, a raje jo pustimo lepo.